Adding Aggressive Early Deflation to the Restructured Symmetric QR Algorithm

Presented by James Levitt

In partial fulfillment of the requirements for graduation with the Dean's Scholars Honors Degree in Computer Sciences

Adding Aggressive Early Deflation to the Restructured Symmetric QR Algorithm

James Levitt
A Dean's Scholars Honors Thesis

Committee:
Dr. Robert van de Geijn
Dr. Alan Cline
Dr. William Press

May 10, 2013

Abstract

The QR algorithm is an algorithm for computing the spectral decomposition of a symmetric matrix [9]. Despite it's high accuracy, other methods are often preferred for the symmetric eigenvalue problem due to the QR algorithm's relatively poor performance [13]. In recent years, new techniques have arisen that dramatically improve its performance. The restructured symmetric QR algorithm, introduced in [13], seeks to increase the rate at which operations are performed, thus reducing execution time. Aggressive early deflation, introduced in [3], seeks to reduce the total number of operations performed. This paper investigates these two techniques and explores the addition of aggressive early deflation to the restructured symmetric QR algorithm.

1 Introduction

The QR algorithm was introduced in the early 1960's as an algorithm for computing the Schur decomposition of a nonysmmetric matrix and the spectral decomposition of a symmetric matrix [9]. In the following thirty-five years, only a few modifications were made to the original algorithm [2]. During that time, other algorithms were introduced such as Cuppen's Divide-and-Conquer method [4] and the method of Multiple Relatively Robust Representations [5] for finding the eigenvalues and eigenvectors of a symmetric tridiagonal matrix. These new methods were found to outperform the symmetric QR algorithm significantly. As a result, many had written off the QR algorithm for the symmetric eigenvalue problem. However, the QR algorithm remained the "method of choice for calculating all eigenvalues and (optionally) eigenvectors of small, nonsymmetric matrices" [2].

Developments in recent years have made the symmetric QR algorithm much more attractive. In 2002, Braman, Byers, and Mathias presented the technique of aggressive early deflation (AED) [3]. The technique was shown to significantly improve the QR algorithm for Hessenberg matrices, but the symmetric case was not discussed. The restructured symmetric QR algorithm [13], developed in 2011, vastly improves efficiency of the symmetric QR algorithm, delivering performance competitive with, though moderately below, the best alternatives when computing the spectral decomposition. Even so, the QR algorithm is arguably preferable because of its accuracy and simplicity, among other advantages.

This paper examines the addition of AED to the restructured symmetric QR algorithm. The aim is to use the technique to gain performance improvement for the restructured symmetric QR algorithm, much as it has already been used for the Hessenberg QR algorithm. All matrices discussed in Sections 2 and 3 are assumed to be real and symmetric unless specified otherwise. Thus, the QR algorithm applied to such matrices is used to find a spectral decomposition.

2 The QR Algorithm for Symmetric Matrices

2.1 The Basic QR Algorithm

The basic QR algorithm is a remarkably simple algorithm for computing the eigenvalue decomposition of a symmetric matrix. As shown in Algorithm 1, each iteration consists of performing a QR factorization and multiplying the factors in reverse order. The sequence of matrices $A^{(k)}$ will converge to a diagonal matrix Λ under suitable assumptions.[1] Since $A^{(k)} = R^{(k)}Q^{(k)} = (Q^{(k)})^T A^{(k-1)} Q^{(k)}$, the matrices $A^{(k)}$ and $A^{(k+1)}$ are similar, and so the iteration preserves eigenvalues. Therefore, the matrix Λ will contain the eigenvalues of A. We can also form the orthogonal matrices $V^{(k)} = Q^{(1)}Q^{(2)} \cdots Q^{(k)}$ so that $A = V^{(k)}A^{(k)}(V^{(k)})^T$. As $A^{(k)}$ converges to the eigenvalues of A, the columns of $V^{(k)}$ converge to the eigenvectors of A.

Algorithm 1 Basic QR Algorithm

```
$A^{(0)} := A$
for $k := 1, \ldots$ do
    $A^{(k-1)} \to Q^{(k)}R^{(k)}$                    $\triangleright$ QR factorization
    $A^{(k)} = R^{(k)}Q^{(k)}$
```

Although elegant, this implementation is very inefficient, with each iteration requiring a QR factorization and multiplication of large, dense matrices at a cost of $\mathcal{O}(n^3)$ per iteration. Fortunately, performance can be improved vastly with a few modifications.

2.2 The "Practical" QR Algorithm

This section introduces three improvements to the basic algorithm aimed at reducing the cost of operations, dividing into problems of smaller size, and increasing the rate of convergence.

2.2.1 Tridiagonal Form

Before iterating, A can be reduced to tridiagonal form with similarity transformations at a one-time cost of $\mathcal{O}(n^3)$. Reduction of A to tridiagonal form can be done by applying a sequence of Householder transformations. Further details regarding tridiagonal reduction can be found in [14, 13, 9, 10]. We define $Q^{(0)}$ to be the similarity transformation that tridiagonalizes A and $A^{(0)}$ to be the resulting tridiagonal matrix. The QR iteration preserves tridiagonal form, so this measure ensures that every $A^{(k)}$ is tridiagonal, reducing the cost of every iteration.

[1]First, the eigenvalues of A must be distinct in absolute value. Second, all principal leading submatrices of V must be nonsingular, where V is the matrix whose columns are the eigenvectors of A in decreasing order of the absolute values of their corresponding eigenvalues [9].

To demonstrate this reduced cost, we examine the QR factorization for tridiagonal matrices. For symmetric, tridiagonal matrix A, the factorization $A = QR$ can be computed by an orthogonal triangularization. That is, we apply orthogonal matrices $G_1^T, ..., G_{n-1}^T$ to A, so that $G_{n-1}^T \cdots G_1^T A = R$. Then we have $A = (G_1 G_2 \cdots G_{n-1})R = QR$. The matrices G_i we use for this task are Givens rotations, which take the form

$$G_i = \left[\begin{array}{c|cc|c} I_{i-1} & & & \\ \hline & \gamma & -\sigma & \\ & \sigma & \gamma & \\ \hline & & & I_{n-i-1} \end{array}\right], \text{ where } \gamma^2 + \sigma^2 = 1.$$

For any vector $x = \begin{pmatrix} \chi_1 \\ \chi_2 \end{pmatrix} \in \mathbb{R}^2$, we can choose $\gamma = \frac{\chi_1}{||x||_2}$ and $\sigma = \frac{\chi_2}{||x||_2}$ so that $\begin{pmatrix} \gamma & -\sigma \\ \sigma & \gamma \end{pmatrix}^T x = \begin{pmatrix} \pm||x||_2 \\ 0 \end{pmatrix}$. The Givens rotation G_i, when applied from the left, will affect rows i and $i+1$ of A. We choose G_1 so that applying it to A will zero the first subdiagonal. Next, we choose G_2 to zero the second subdiagonal of $G_1^T A$, and so on. Applying these Givens rotations will also introduce nonzeros above the diagonal, which is fine as long as the final result is upper triangular. After applying $n-1$ Givens rotations we have $Q = G_1 G_2 \cdots G_{n-1}$ and $R = G_{n-1}^T \cdots G_1^T A$.

The formation and application of each G_i requires a constant number of operations, so the QR factorization for symmetric, tridiagonal A takes $\mathcal{O}(n)$ time. It is worth noting that in order to complete an iteration of the QR algorithm, we would need to apply the Givens rotations to R from the right and also apply them to the matrix of eigenvectors $V^{(k)}$, which takes $\mathcal{O}(n^2)$ time. The QR algorithm in Section 2.3 abandons the QR factorization, but the technique of sweeping through a matrix with Givens rotations remains very important.

2.2.2 Deflation

Next, whenever an iteration produces a matrix with some offdiagonal entry sufficiently close to zero, the problem can be "deflated." That is, the offdiagonal entry is set to zero, resulting in a matrix of the form $\left[\begin{smallmatrix} A_1 & 0 \\ 0 & A_2 \end{smallmatrix}\right]$ with tridiagonal blocks A_1, A_2, and then the QR algorithm is applied separately to A_1 and A_2. Deflation is often due to the last offdiagonal element being sufficiently close to zero. In this case, the single element of A_2 is an eigenvalue of A, and the QR algorithm continues only on A_1.

2.2.3 Shifts

The idea behind the QR algorithm with shifts is that the QR iteration can be performed on a shifted matrix (with shifted eigenvalues) that will converge to eigenvalues much faster. The shift μ is chosen in a way to approximate an eigenvalue of A. The QR factorization step is then performed on $A^{(k-1)} - \mu^{(k)}I$, and the shift is later undone by adding $\mu^{(k)}I$ to the product $R^{(k)}Q^{(k)}$. The details of how to determine a shift and why this shifting strategy works can be found in [14, 9, 10].

Incorporating the initial tridiagonal reduction, deflation, and a shifting strategy produces Algorithm 2, referred to as the "Practical" QR Algorithm [9].

Algorithm 2 "Practical" QR Algorithm

```
$(Q^{(0)})^T A^{(0)} Q^{(0)} := A$                    ▷ Tridiagonalize $A$
$V^{(0)} := Q^{(0)}$
for $k := 1, ...$ do
    $\mu^{(k)} :=$ some shift
    $A^{(k-1)} - \mu^{(k)}I \rightarrow Q^{(k)}R^{(k)}$        ▷ QR factorization
    $A^{(k)} = R^{(k)}Q^{(k)} + \mu^{(k)}I$
    $V^{(k)} := V^{(k-1)}Q^{(k)}$                ▷ Update eigenvector matrix $V$
    Deflate when possible
```

2.3 The QR Algorithm with Implicit Shifts

The QR Algorithm with Implicit Shifts introduces a further improvement that replaces the QR factorization entirely, performing computation equivalent to the shifted QR algorithm, but more efficiently.

2.3.1 The Francis QR Step

We start by choosing a shift μ, as previously discussed. The Givens rotation G_1 is formed such that applying G_1 to $A - \mu I$ zeros the first subdiagonal entry. Then, G_1 is applied as a similarity transform to (unshifted) A. This affects the first two rows and columns, creating a nonzero element below the subdiagonal and a symmetric counterpart above the superdiagonal, which together are referred to as the "bulge." The result $G_1^T A G_1$ is shown below, with the '$\star$' symbol representing non-zero entries.

$$G_1^T A G_1 = \begin{bmatrix} \star & \star & \star & & & & & \\ \star & \star & \star & & & & & \\ \star & \star & \star & \star & & & & \\ & & \star & \star & \star & & & \\ & & & \star & \star & \star & & \\ & & & & \star & \star & \ddots & \\ & & & & & \ddots & \ddots & \star \\ & & & & & & \star & \star \end{bmatrix}$$

Now, we begin the process known as chasing the bulge, which has some similarities to the process of zeroing successive subdiagonal entries with Givens rotations discussed in Section 2.2.1. The Givens rotation G_2 is formed, which acts on the second and third rows/columns to zero the bulge. Applying G_2 as a similarity transformation, we get a matrix of the form

$$G_2^T G_1^T A G_1 G_2 = \begin{bmatrix} \star & \star & & & & & & \\ \star & \star & \star & \star & & & & \\ & \star & \star & \star & & & & \\ & \star & \star & \star & \star & & & \\ & & & \star & \star & \star & & \\ & & & & \star & \star & \ddots & \\ & & & & & \ddots & \ddots & \star \\ & & & & & & \star & \star \end{bmatrix}.$$

As shown, G_2 has modified entries in the second row and column to create another bulge. We continue this process, forming $G_3, ..., G_{n-1}$ to eliminate the bulge left by the previous Givens rotation. This process continues with the bulge moving downward along the offdiagonals until G_{n-1} finally restores tridiagonal form. One such sweep, consisting of introducing the bulge and chasing it all the way through the matrix, is known as a Francis step.

The Francis step can be shown by the Implicit Q Theorem to be equivalent to one step of the shifted QR algorithm [14, 10]. The Francis step performs the work of both the QR factorization and the multiplication of the factors R and Q, at a much lower cost.

2.4 The Restructured Symmetric QR Algorithm

The restructured symmetric QR algorithm presented in [13] is motivated by the observation that the Classic QR Algorithm is rich in operations that perform $\mathcal{O}(n^2)$ computations on $\mathcal{O}(n^2)$ data. The rate of execution of such computations can be bottlenecked by memory accesses. It was found that existing implementations of the symmetric QR algorithm could benefit significantly from more efficient memory access. The restructured algorithm achieved improved performance by reducing the number of memory operations performed and increasing the reuse of cached data. The actual computation performed is exactly the same, but memory accesses are now $\mathcal{O}(n^2)$ relative to $\mathcal{O}(n^2k)$ computations, where k is some tunable parameter.

2.4.1 Accumulating Sets of Givens Rotations

The practice of updating the $n \times n$ matrix of eigenvectors V by applying one set of $(n-1)$ Givens rotations at a time makes inefficient use of costly memory operations. One such step performs $6n$ flops per Givens rotation for a total of nearly $6n^2$ flops for the full set. If each element of V is read and written only once, this step requires $2n^2$ memory operations at the minimum. Therefore, this step can be performed with a maximum ratio of 3 flops per memory operation. In practice, however, LAPACK[2] routines for applying Givens rotations achieve only 1.5 flops per memory operation. If, instead, k sets of Givens rotations are accumulated, the step of applying all of these rotations performs about $6kn^2$ flops. The minimum number of memory operations remains $2n^2$, for a much more favorable theoretical maximum ratio of $3k$ flops per memory operation.

This potential improvement is captured by rearranging the order in which Givens rotations are applied. Figure 1 represents four sets of Givens rotations, with each rotation positioned above the two columns of V to which it is applied. The arrows represent dependencies that restrict the order in which the rotations can be applied. Taking the Givens rotations as vertices

[2]Linear Algebra PACKage (LAPACK) is a software package written in Fortran that provides routines for performing various numerical linear algebra computations, including an implementation of the QR algorithm [1].

$$\begin{tikzcd}
G_{0,3} \arrow[r] & G_{1,3} \arrow[r] & G_{2,3} \arrow[r] & G_{3,3} \arrow[r] & G_{4,3} \arrow[r] & G_{5,3} \arrow[r] & G_{6,3} \arrow[r] & G_{7,3} \arrow[r] & G_{8,3} \\
G_{0,2} \arrow[r] \arrow[u] & G_{1,2} \arrow[r] \arrow[u] \arrow[ul] & G_{2,2} \arrow[r] \arrow[u] \arrow[ul] & G_{3,2} \arrow[r] \arrow[u] \arrow[ul] & G_{4,2} \arrow[r] \arrow[u] \arrow[ul] & G_{5,2} \arrow[r] \arrow[u] \arrow[ul] & G_{6,2} \arrow[r] \arrow[u] \arrow[ul] & G_{7,2} \arrow[r] \arrow[u] \arrow[ul] & G_{8,2} \arrow[u] \arrow[ul] \\
G_{0,1} \arrow[r] \arrow[u] & G_{1,1} \arrow[r] \arrow[u] \arrow[ul] & G_{2,1} \arrow[r] \arrow[u] \arrow[ul] & G_{3,1} \arrow[r] \arrow[u] \arrow[ul] & G_{4,1} \arrow[r] \arrow[u] \arrow[ul] & G_{5,1} \arrow[r] \arrow[u] \arrow[ul] & G_{6,1} \arrow[r] \arrow[u] \arrow[ul] & G_{7,1} \arrow[r] \arrow[u] \arrow[ul] & G_{8,1} \arrow[u] \arrow[ul] \\
G_{0,0} \arrow[r] \arrow[u] & G_{1,0} \arrow[r] \arrow[u] \arrow[ul] & G_{2,0} \arrow[r] \arrow[u] \arrow[ul] & G_{3,0} \arrow[r] \arrow[u] \arrow[ul] & G_{4,0} \arrow[r] \arrow[u] \arrow[ul] & G_{5,0} \arrow[r] \arrow[u] \arrow[ul] & G_{6,0} \arrow[r] \arrow[u] \arrow[ul] & G_{7,0} \arrow[r] \arrow[u] \arrow[ul] & G_{8,0} \arrow[u] \arrow[ul] \\
v_0 & v_1 & v_2 & v_3 & v_4 & v_5 & v_6 & v_7 & v_8 & v_9
\end{tikzcd}$$

Figure 1: A representation of four sets of Givens rotations. Givens rotation $G_{i,j}$ is the i^{th} Givens rotation of the j^{th} set, which is applied to columns v_i, v_{i+1} of V. The rotation at the tail of an arrow must be applied before the rotation at the head of the arrow is applied. This figure is reproduced from [13].

and the arrows as edges forms a directed acyclic graph such that any topological ordering of the graph gives an order of applying Givens rotations that is equivalent to applying the rotations sequentially in the classic ordering.

The dotted lines in the figure partition the Givens rotations into "waves." In the restructured algorithm, Givens rotations within a wave are applied consecutively from bottom to top, and waves are applied one after another from bottom-left to top-right. This pattern makes extensive use of data already residing in the cache, with an average of about one column of V accessed from memory per wave of Givens rotations applied.

The number k of sets of Givens rotations to be applied together is limited by the size of the cache. In particular, the cache must be able to hold the $k + 1$ columns of V to which a wave of k Givens rotations are applied. This presents a problem for large matrices where very few columns can be held in cache. To deal with such matrices, V is blocked by rows, and Givens rotations are applied to each block separately. Clearly, there is a tradeoff between larger k, which allows for more flops per step, and larger blocking size of V, which results in fewer sweeps through the accumulated Givens rotations.

2.4.2 Performance of the Restructured Symmetric QR Algorithm

Figure 2 shows performance of various methods for applying Givens rotations. The highest performance is attained by the application of multiple sets of Givens rotations with 2×2 fusing, a further optimization described in [13]. Also shown is the maximum attainable peak for the application of Givens rotations, which is 25% less than the machine's peak. This is due to the processor architecture, which is capable of executing one multiplication and one addition simultaneously. The application of Given's rotations requires two multiplications for every addition, effectively leaving one in four available flops unutilized. Given this limitation, the restructured algorithm performs this operation very efficiently. Its performance relative to peak is similar to that of dense matrix-matrix multiplication, suggesting that the restructured algorithm remedies the memory-related performance issues associated with the application of Givens rotations.

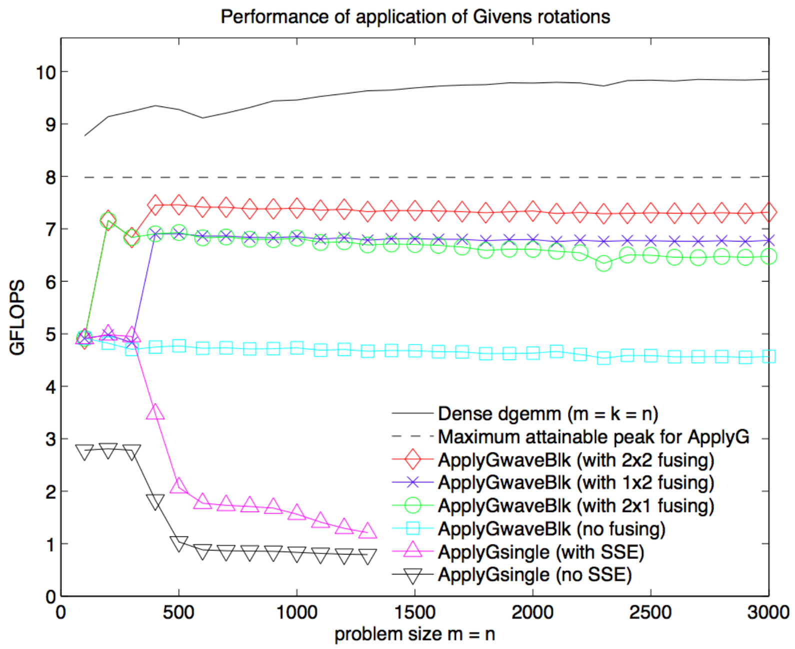

Figure 2: A graph showing performance of applying $k = 192$ sets of Givens rotations with implementations that apply multiple sets in waves (with various levels of fusing) and those that apply one set at a time. Implementations that use blocking were run with a blocksize of 256. The experiment was performed on a single core of a six-core Intel "Dunnington" processor with a peak performance of 10.64 GFLOPS. This figure is reproduced from [13].

3 Aggressive Early Deflation

The strategy for deflation presented in Section 2.1 involves searching the offdiagonal for entries that are sufficiently close to zero, setting those entries to zero, and continuing the QR algorithm separately on the resulting submatices. This technique is useful for detecting converged eigenvalues and reducing the size of the problem, thereby reducing the number and cost of QR iterations performed. However, it does not always detect all opportunities for deflation. The technique of aggressive early deflation (AED) introduced in [3] is a strategy for detecting opportunities for deflation that are not found by inspection of the offdiagonal.

3.1 The AED Step

The AED step first requires computing the eigenvalue decomposition of the "deflation window," a $b \times b$ trailing submatrix of the symmetric tridiagonal $n \times n$ matrix T. To do so, T is partitioned as shown:

$$T \to \left[\begin{array}{c|c} T_{11} & T_{12} \\ \hline T_{21} & T_{22} \end{array}\right] = \left[\begin{array}{cccc|cccc} d_1 & e_1 & & & & & & \\ e_1 & \ddots & \ddots & & & & & \\ & \ddots & \ddots & e_{n-b-1} & & & & \\ & & e_{n-b-1} & d_{n-b} & e_{n-b} & & & \\ \hline & & & e_{n-b} & d_{n-b+1} & e_{n-b+1} & & \\ & & & & e_{n-b+1} & \ddots & \ddots & \\ & & & & & \ddots & \ddots & e_{n-1} \\ & & & & & & e_{n-1} & d_n \end{array}\right]$$

The eigenvalue decomposition $T_{22} = Q_{22}\Lambda_{22}Q_{22}^T$ is computed, which can be done, for example, with the restructured symmetric QR algorithm. Next, Q_{22} and the $(n-b) \times (n-b)$ identity matrix I_{n-b} are used to construct the orthogonal matrix Q_{AED}:

$$Q_{AED} = \begin{bmatrix} I_{n-b} & 0 \\ 0 & Q_{22} \end{bmatrix}$$

Performing the similarity transformation $Q_{AED}^T T Q_{AED}$ in blocks gives

$$Q_{AED}^T T Q_{AED} = \begin{bmatrix} T_{11} & T_{12}Q_{22} \\ Q_{22}^T T_{21} & \Lambda_{22} \end{bmatrix}$$

Note that the columns $Q_{22}^T T_{21}$ are all zero except for the last one, which is equal to e_{n-b} times the first column of Q_{22}^T. Also, since symmetry is preseved, the rows of $T_{12}Q_{22} = (Q_{22}^T T_{21})^T$ are all zero except for the last one, which is equal to e_{n-b} times the first row of Q_{22}. This column of $Q_{22}^T T_{21}$ and row of $T_{12}Q_{22}$ are referred to as the "spike." The second stage of Figure 3 represents the matrix after the spike has been formed.

Next, we inspect the spike for elements that are sufficiently close to zero, and set them to zero. These zeroed spike elements are represented in the third stage of Figure 3 by white boxes. If such elements exist, then the corresponding elements on the diagonal, shown in the figure as green boxes, are converged eigenvalues. So, we form a permutation matrix P, that when applied from the right will permute the columns of $Q_{AED}^T T Q_{AED}$ to move the zeroed elements to the end of the horizontal spike. Similarly, applying P^T from the left will have the same effect on the vertical spike. The fourth stage of Figure 3 represents the result of the similarity transformation

$$P^T Q_{AED}^T T Q_{AED} P.$$

Now, the converged eigenvalues are located on the trailing diagonal, and the next step is to restore tridiagonal form. So, we perform the tridiagonalization

$$Q_{tri}^T P^T Q_{AED}^T T Q_{AED} P Q_{tri} = T'$$

From T', we deflate the converged eigenvalues, and continue with the deflated problem, as shown in the final stage of Figure 3. The last remaining task is to update the matrix of eigenvectors V. If the matrix of eigenvectors prior to performing the AED step is $V^{(k)}$, then we form the matrix $V^{(k+1)}$ by

$$V^{(k+1)} = V^{(k)} Q_{AED} P Q_{tri},$$

so that

$$V^{(k+1)} T' (V^{(k+1)})^T = V^{(k)} T (V^{(k)})^T = A.$$

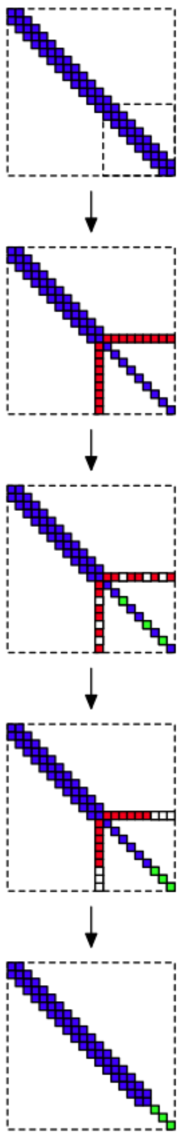

Figure 3: A graphical representation of one step of the aggressive early deflation on a symmetric tridiagonal matrix. This figure is reproduced from [7].

3.2 Discussion of AED

Effect on performance The purpose of AED is to improve performance by reducing the number of QR iterations performed. In order to do so, the AED step performs multiple $\mathcal{O}(b^3)$ operations, including a full eigenvalue decomposition and tridiagonalization. The key is that the size b of the deflation window is chosen to be small compared with T. For small enough values of b, the cost of an AED iteration is much less than that of a QR iteration, which sweeps through the entirety of T. As a result, the use of aggressive early deflation can improve performance.

Zeroing elements on the spike The criterion for determining when to zero an element on the spike must aim for frequent deflations while maintaining accuracy. Several possible criteria are discussed in [3] for the nonsymmetric case, but no definitive answer is given as to which is best, and the symmetric case was not specifically addressed. Additional examples of zeroing criteria are given in [8] for use in computing the singular value decomposition of a symmetric matrix.

Multiple applications of AED One strategy involves making multiple consecutive attempts at aggressive early deflation. After a successful deflation we may apply the technique again on the deflated problem. It may be worthwhile to attempt AED again before the next QR sweep. This raises the question of how many times AED should be applied before switching back to the outer QR iteration. One possibility, presented in [6], is to continue performing AED steps as long as the ratio of the number of deflations to the size of the deflation window exceeds some machine-dependent threshold. Also, it may be possible to deflate the problem at a lower cost by using multiple applications of AED on a small window rather than a single application on a large window.

4 Opportunities

The restructured symmetric QR algorithm provides a natural point at which to attempt aggressive early deflation. After a set of Givens rotations has been accumulated and applied, and before work has begun on the next set, we have a tridiagonal matrix T along with an updated matrix of eigenvectors

V. At this point, aggressive early deflation is attempted, and T and V are updated if deflation occurs.

Implementations of aggressive early deflation for other problems have seen the greatest benefit from applying the technique frequently - in some cases, more than once per Francis step. This is not feasible for the restructured symmetric QR algorithm, where performance relies on accumulating and applying multiple sets of Givens rotations at a time. When adding AED, it may be possible to benefit by reducing the number of accumulated sets of Givens rotations, effectively trading some performance in applying Givens rotations for more frequent attempts at aggressive early deflation. On the other hand, it is extremely important to keep a large enough number of accumulated sets of Givens rotations in order to maintain the Level-3 performance of applying those sets.

The size of the deflation window depends on the problem size and the size of the cache. The size of the deflation window must be small enough relative to the size of the problem that attempting aggressive early deflation is inexpensive. Also, the deflation window size should be chosen so that all of the data required for completing an AED attempt can fit in the cache. Thus, theses factors give constraints on the maximum size of the deflation window. Within these limits, better performance was observed when using a larger deflation window.

It is possible to use a deflation window of varying size for each AED attempt. As the problem deflates and splits into smaller subproblems, we can choose the deflation window to fit the size of the problem at hand. Varying the size of the deflation window was observed to provide some small benefit, but it was not much better than using a carefully chosen fixed size. The vast majority of the time is spent working on the large problem, and the smaller problems that arise from deflations and splits are solved relatively quickly. As a consequence of Amdahl's law, speedup for the small problems have little impact on overall performance.

The strategy of performing multiple consecutive AED attempts was observed to be somewhat successful for a small deflation window, but better performance was achieved by using a large window with only one attempt at a time. This may be due to the fact that a larger window can converge eigenvalues higher up on the diagonal when a smaller window might not converge any. However, this observation might not hold for a different architecture. For instance, if the cache is small enough relative to the problem size, it may be the case that even the largest window allowed by the cache constraint is small enough that multiple AED attempts are worthwhile.

We must also choose when to attempt aggressive early deflation. This was observed to depend on the size of the deflation window. Larger windows tend to be most effective after more sets of Givens rotations have been applied. Viewing AED as introducing a deflating perturbation, it makes sense that we must allow QR sweeps to make some progress between AED attempts. This can be done by increasing the number of accumulated sets of Givens rotations (which is limited by the cache size) or by simply skipping opportunities to attempt AED.

5 Results

Aggressive early deflation was added to existing implementations of the restructured QR algorithm in the `libflame` linear algebra library [12, 11]. The technique was added to two variants of the restructured QR algorithm, referred to as rQR and rQR II. The implementations with AED added are referred to as rQR+AED and rQR II+AED. For comparison, an implementation of the original QR algorithm as implemented in netlib LAPACK is included as oQR.

Experiments were performed on a four-core 2.66 GHz Intel Xeon X5355 running Linux 2.6.32-45-server. The experiments were run on complex Hermitian matrices with double-precision floating point arithmetic that are already tridiagonalized. Thus, performance is shown for the tridiagonal eigenvalue decomposition and formation of eigenvectors.

Figure 4 shows performance of each implementation relative to oQR on an input matrix with eigenvalues following a geometric distribution, and using a fixed deflation window size of 256. As discussed in the previous section, the size of the deflation window may vary and AED may be skipped. The results when using a deflation window of size 256 are useful for observing which problem sizes benefit from such application of AED. In general, smaller windows offer better performance for smaller matrices but with limited improvement for larger matrices. On the other hand, larger windows are worse for small matrices but offer a higher speedup factor for large matrices.

For this experiment, the addition of AED was seen to roughly double performace of rQR and improve performance of rQR II by about 50% for certain problem sizes. Figures 5 and 6 in the appendix show performance for matrices with eigenvalues following other distributions.

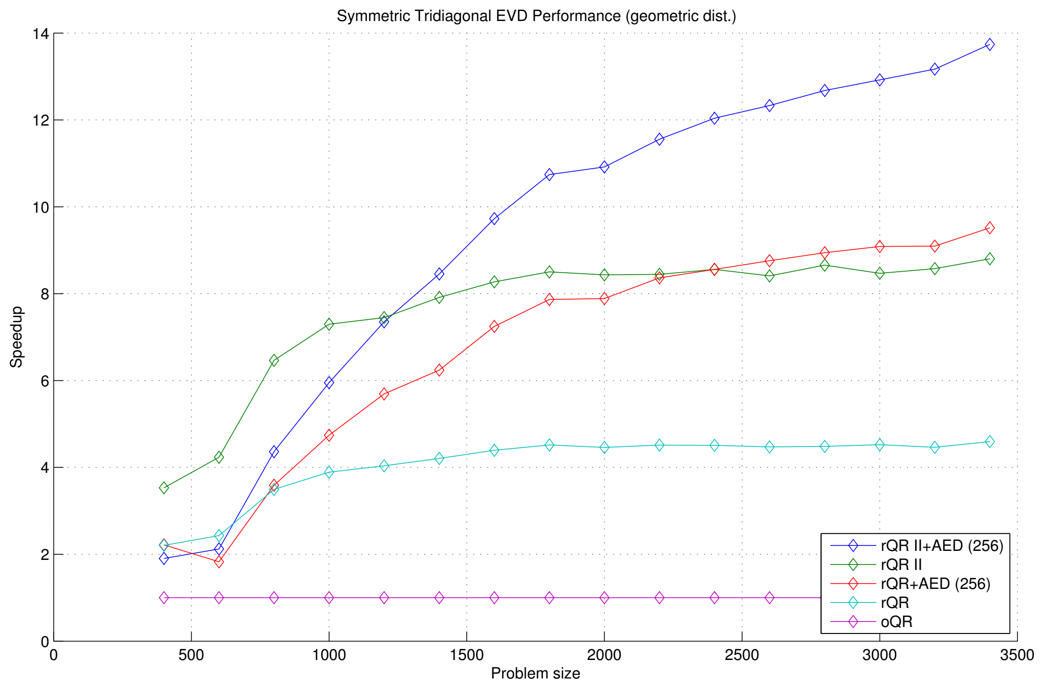

Figure 4: Performance of various implementations relative to oQR on a matrix with eigenvalues following a geometric distribution. For implementations that use AED, the deflation window size was set to 256.

6 Conclusions

The QR algorithm has long been an important tool for solving the eigenvalue problem, yet other algorithms with high-performance implementations have been preferred for the symmetric eigenvalue problem. This paper presents the QR algorithm and two recent advancements that improve its performance, and explores the addition of the technique of aggressive early deflation to the restructured symmetric QR algorithm. The results show that adding aggressive early deflation further improves the performance of the restructured QR algorithm, bringing closer within range of the fastest symmetric eigensolvers. It is plausible that further work on this subject will yield additional performance improvement. As noted in [13], the QR algorithm carries other positive features to consider including high accuracy and simplicity, among others. With the improvement demonstrated in this paper, the QR algorithm becomes even more appealing as a method for solving the symmetric eigenvalue problem.

7 Acknowledgements

I would like to thank Dr. Robert van de Geijn for his guidance and encouragement throughout this research project. I also thank Field Van Zee for his work on the `libflame` library including the restructured QR implementations to which I added AED, and for answering my questions regarding `libflame`. I credit Meiyue Shao for his suggestion on the criterion for zeroing elements on the spike. Finally, I thank my co-advisor Dr. Alan Cline and committee member Dr. William Press for their input.

This work was partially sponsored by the NSF (Grant OCI-1148125).

Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation (NSF).

References

[1] E. Anderson, Z. Bai, C. Bischof, L. S. Blackford, J. Demmel, Jack J. Dongarra, J. Du Croz, S. Hammarling, A. Greenbaum, A. McKenney, and D. Sorensen. *LAPACK Users' Guide (third ed.)*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 1999.

[2] Karen Braman, Ralph Byers, and Roy Mathias. The multishift QR algorithm. part I: Maintaining well-focused shifts and level 3 performance. *SIAM J. Matrix Anal. Appl.*, 23:929–947, April 2001.

[3] Karen Braman, Ralph Byers, and Roy Mathias. The multishift QR algorithm. part II: Aggressive early deflation. *SIAM J. Matrix Anal. Appl.*, 23:948–973, April 2001.

[4] J. J. M. Cuppen. A divide and conquer method for the symmetric tridiagonal eigenproblem. *Numerische Mathematik*, 36:177–195, 1980.

[5] Inderjit Singh Dhillon. *A New $O(n^2)$ Algorithm for the Symmetric Tridiagonal Eigenvalue/Eigenvector Problem.* PhD thesis, EECS Department, University of California, Berkeley, Oct 1997.

[6] Robert Granat, Bo Kågström, Daniel Kressner, and Meiyue Shao. Parallel library software for the multishift QR algorithm with aggressive early deflation. Technical Report UMINF 12.06, Umeå University, 2012.

[7] Daniel Kressner. Semester/master project: Aggressive early deflation for the symmetric QR algorithm. Unpublished manuscript.

[8] Y. Nakatsukasa, K. Aishima, and I. Yamazaki. dqds with aggressive early deflation. *SIAM J. Matrix Anal. Appl.*, 33(1):22–51, 2012.

[9] Lloyd N. Trefethen and David Bau. *Numerical Linear Algebra.* SIAM: Society for Industrial and Applied Mathematics, 1997.

[10] Robert A. van de Geijn. Notes on the symmetric QR algorithm. `http://www.cs.utexas.edu/users/flame/Notes/NotesOnSymQR.pdf`, 2010.

[11] Field G. Van Zee. `libflame`*: The Complete Reference.* `www.lulu.com`, 2012.

[12] Field G. Van Zee, Ernie Chan, Robert van de Geijn, Enrique S. Quintana-Ortí, and Gregorio Quintana-Ortí. The libflame library for dense matrix computations. *IEEE Computation in Science & Engineering*, 11(6):56–62, 2009.

[13] Field G. Van Zee, Robert van de Geijn, and Gregorio Quintana-Ortí. Restructuring the QR algorithm for high-performance application of Givens rotations. Technical Report TR-11-36, The University of Texas at Austin, Department of Computer Science, October 2011. FLAME Working Note #60.

[14] David S. Watkins. *Fundamentals of Matrix Computations.* Wiley, Newark, NJ, 1991.

A Additional Performance Figures

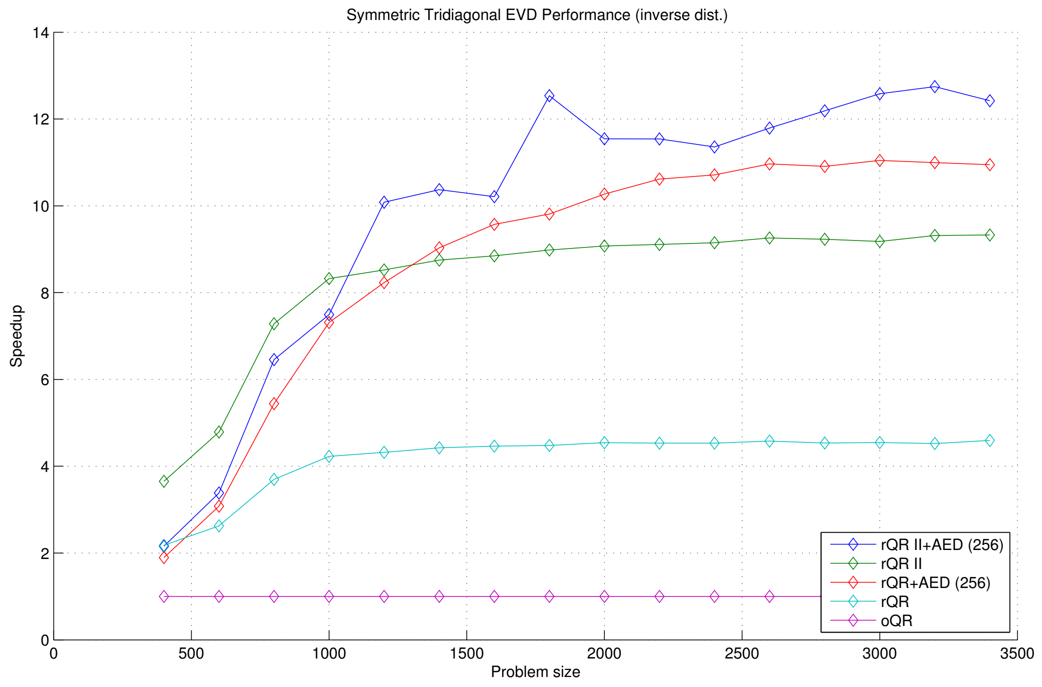

Figure 5: Performance of various implementations relative to oQR on a matrix with eigenvalues following an inverse distribution. For implementations that use AED, the deflation window size was set to 256.

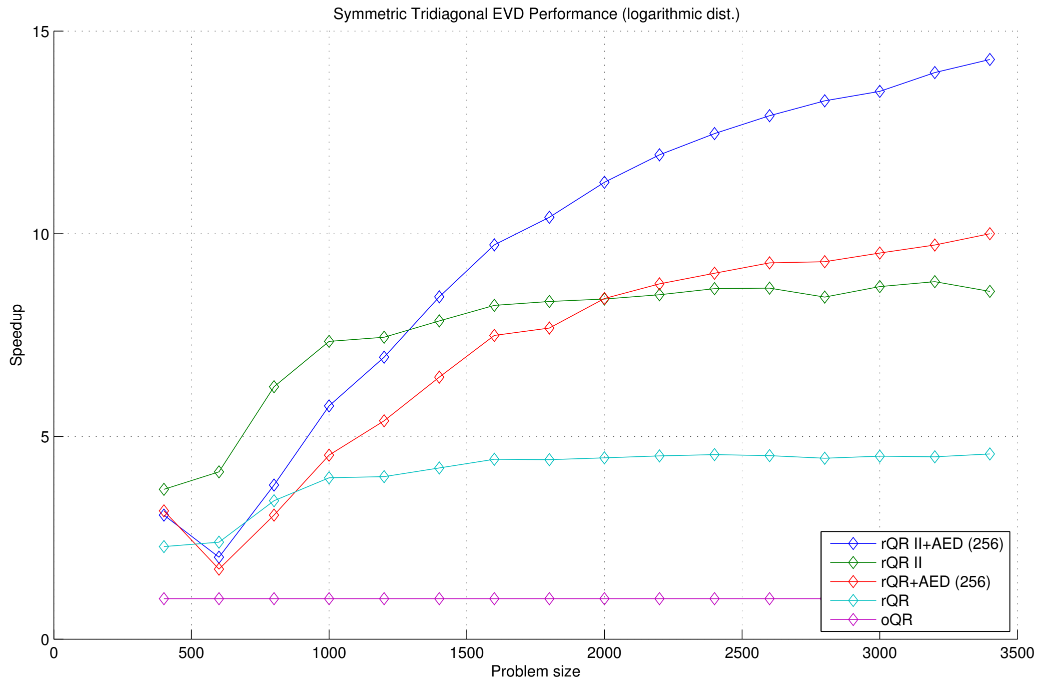

Figure 6: Performance of various implementations relative to oQR on a matrix with eigenvalues following a logarithmic distribution. For implementations that use AED, the deflation window size was set to 256.